The George Washington University
WASHINGTON DC

OFFICE OF SPONSORED RESEARCH

May 20, 1997

Charles K. Hayes
Administrative Contracting Officer
Department of Navy
Atlanta Regional Office
100 Alabama Street, N.W.
Suite 4R15
Atlanta, GA 30303-3104

Re: Contract No. N00014-94-K-2009; Final Technical Report

Dear Mr. Hayes:

The George Washington University recently submitted a letter dated May 16, 1997 regarding the close out of the contract mentioned above. The final technical report included with this letter was the incorrect report, please discard. Here is the correct final report.

We apologize for any inconvenience this might have caused you.

Sincerely,

Gianna Rudolph
Post Award Coordinator



enclosure



cc: Nick Kaplan
James Clements
Director, NRL
Defense Technical Information Center

2121 EYE STREET, N.W. • WASHINGTON, DC 20052 • (202) 994-6255 • FAX (202) 994-9137

# The Virtual Audio Server

**Hesham Fouad and James Hahn**

**Department of Electrical Engineering and Computer Science**

**The George Washington University**

**Washington, DC 20052**

## Abstract

Sound is certain to become a standard component of Virtual Environment (VE) interfaces. Current technology, however, has not brought about the wide spread use of sound in VEs. Research efforts have focused primarily on the localization problem - making sounds appear to emanate from a particular direction in 3D space. The problems of modeling the sonic environment and sound generation have not been adequately addressed.

This paper presents the Virtual Audio Server (VAS), a system designed to address the problems of integrating sound into VEs. VAS is a real-time, distributed sound server. It provides high level abstractions for modeling the sonic environment. VAS's extensible architecture allows it to support a variety of rendering techniques. Parameterizable synthetic sound source are supported and a novel graceful degradation technique manages the real-time generation of those sound sources.

## 1. Introduction

Virtual Environment (VE) interfaces are meant to immerse users in interactive computer generated worlds. The information content that a VE can convey to those users depends largely on the sensory channels utilized. While emphasis has historically been on the visual channel, researchers are beginning to recognize the importance of sound as a tool for conveying information to users and for enhancing the immersive qualities of VEs.

Research efforts in sounding VEs have focused primarily on techniques for localizing sounds. While this is an important problem, it is certainly not the whole picture. Three basic problem areas need to be addressed by a VE sound system (fig. 1):

**Modeling the sonic environment.** Modeling abstractions describe the static and dynamic properties of the elements comprising a sonic environment. Those abstractions should provide a rich programming toolkit for describing the environment while accommodating varying levels of expertise by the programmer. Beyond that, modeling abstractions should facilitate the

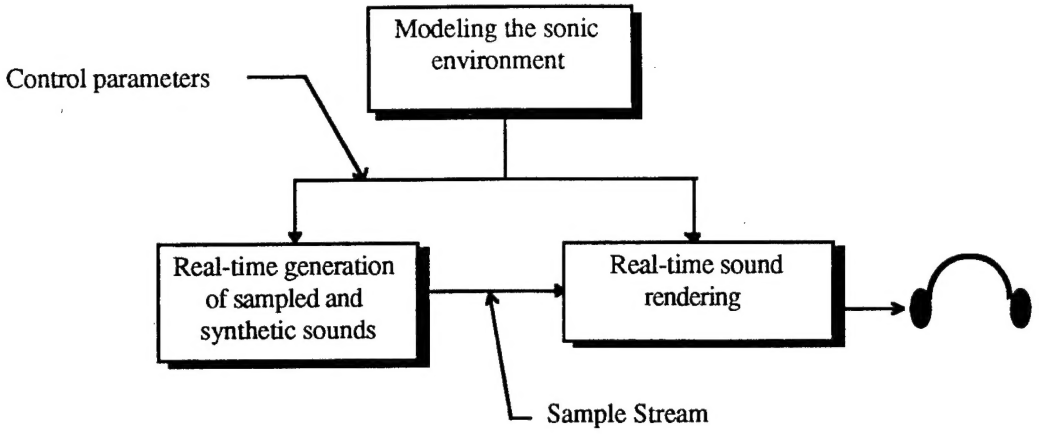


**Figure 1:** Spatial sound generation for VE

mapping of elements in the sonic environment to graphical constructs forming the visual environment. This helps the programmer to establish a coherent audio-visual environment.

**Real-time sound generation.** Sound generation is the process of evaluating the representation of a sound at fixed intervals in order to produce an audio sample stream. Given that the number of concurrently active sound sources in an environment is not known a priori, it is possible that a point is reached where the available computational resources are not sufficient to maintain real-time evaluation of the sample streams. The sound generation mechanism must manage the computational resources so that the perceptible effects of an overload condition are minimized.

**Real-time sound rendering.** Rendering sounds entails two problems: localization and simulating the environmental effects. Localization is the process of recreating spatial auditory cues so that sounds appear to emanate from a particular direction in 3D space. This can be done by recreating the sound sources around the listener using an array of strategically placed loudspeakers or by using filter convolution to simulate Head Related Transfer Functions [3] (HRTFs) and replaying the resultant sound over headphones. The suitability of these two techniques depends largely on the intended application. Calculating environmental effects requires that sound waves be traced from source to listener taking into account reflection, diffraction, and attenuation. Little work has been done in developing real-time techniques for simulating those effects. A VE sound system should support a variety of existing rendering techniques and also facilitate the exploration of new ones.

There has been some work done in addressing the problem of modeling the sonic environment. The *Audition* sound server [1] uses an actor paradigm to model sounds and provides constructs for controlling and synchronizing them in real-time. The ENO server [19] is intended for incorporating non-speech audio cues into Unix applications. It introduces a novel approach to representing sounds using higher level abstractions such as sources, interactions, attributes and sound spaces. Work has also been done in sound synchronization and rendering in computer animation [9, 10].

Research efforts in addressing the localization problem have resulted in a number of approaches. The *NA3* audio server [2] incorporates multiprocessing techniques in order to support software-based localization using HRTFs. The *Acoustitron* [4], a commercially available system developed by Crystal River Engineering, utilizes DSP coprocessors to provide hardware-based HRTF localization as well as simulation of some environmental effects. Finally, the *NPSNET-PAS* server [5] uses MIDI to control a sample playback device in order to generate sounds. Sounds are localized using six speakers surrounding the listener. A similar approach is used by *Personal Audio* [6], a commercially available system developed by VSI.

The main problem with these approaches is that they focus primarily on localization, and do not address the whole process of integrating sound into VEs. The lack of resource management for real-time sound generation, for example, forces programmers to either limit the number of sound sources they use or to devise ad hoc methods for prioritizing sounds in the environment. Modeling abstractions in these systems are geared primarily to describing the physical aspects of the sonic environment without taking into account the visual constructs with which they must be associated. Finally, current systems are "hardwired" to use a single localization technique which may not be ideal for all the applications.

This paper describes the Virtual Audio Server (VAS) which addresses the problems described above. VAS provides the programmer with high level abstractions for modeling the sonic environment, requiring little or no knowledge of underlying audio hardware. The system supports both sampled and synthetic representations of sound so that a wide range of options are

available in determining the content of the sonic environment. Sound generation is managed by a real-time scheduler that provides graceful degradation in the case of overload. An extendible architecture allows the system to support a variety of existing as well as future rendering techniques. Finally, in order to promote the use of sound in VEs, VAS has been made publicly available on the internet1. The system has been used to integrate sound into a number of large scale VE systems.

In the following sections we describe the salient features of the VAS system. Section 2 presents an overview the system. Section 3 discusses how VAS models the sonic environment. In Section 4 we discuss problems with real-time sound generation. Finally in section 5, we discuss VAS's support for various real-time sound rendering techniques.

## 2. System Overview

1 VAS is available via anonymous ftp from ftp.gwu.edu/VAS

VAS is partitioned into four functional areas as depicted in fig. 2. *Remote Objects* provide client applications with access to the server. Each server object with which the client interacts has a corresponding Remote Object on the client's machine. This object acts as a local representative, mirroring the state of its server object and communicating with that object when necessary. This approach maintains an object oriented interface to the server and minimizes the communication required between client and server.

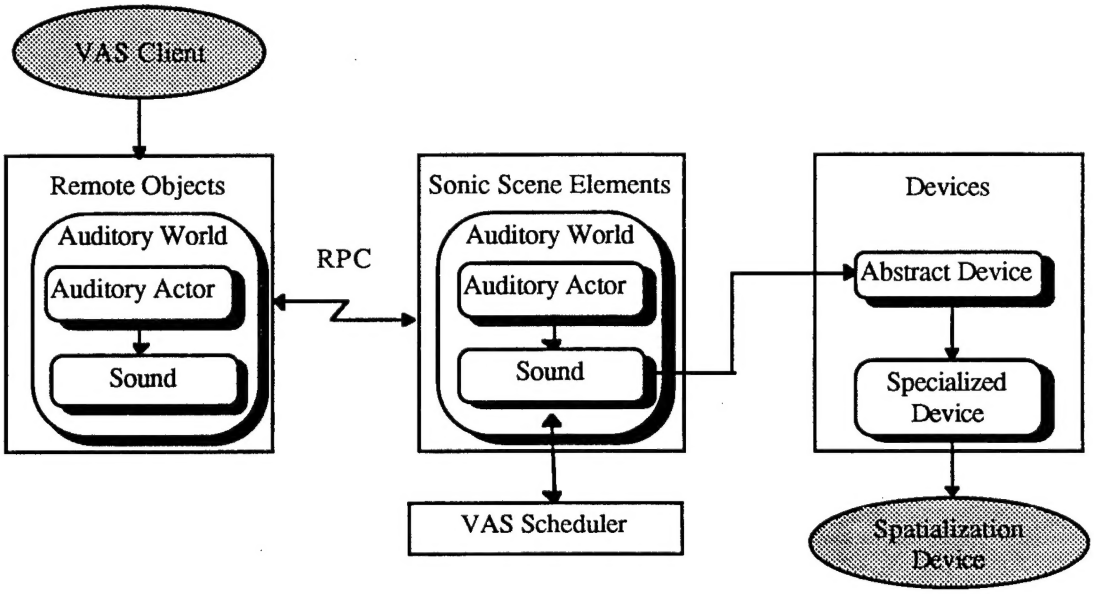


**Figure 2:** VAS system architecture

*Sonic Scene Elements* model the sonic environment. They consist of the *Auditory World* which maintains the state of the sonic environment including all the objects within it. *Auditory Actors* model high level scene elements which consist of listeners, spaces, and sound producing entities. *Sounds* evaluate sound samples and write the resultant samples to *Devices*. An instance of a Device is attached to each Sound and provides it with a device independent interface to any rendering mechanism used by the server. Finally, the *Scheduler* manages the real-time evaluation of active sounds in the environment.

# 3. Modeling the Sonic Environment

VAS incorporates an actor based model of the sonic environment which closely parallels many of the constructs developed for modeling the visual environment. This facilitates the integration process because the two models naturally correspond and their constituent objects are functionally similar.

## 3.1. Auditory Actors

VAS models any sound producing entity in the world using the Auditory Actor. Auditory Actors give the visual entities in the scene auditory properties in the form of a sound repertoire. The Auditory Actors' interface provides control primitives for positioning the actor in 3D space, adding and removing sounds from the actor's sound repertoire, for controlling the play state of those sounds and for controlling a sound's timbre through its parameter space.

Auditory Actors can be controlled directly using the primitives described above, or their actions can be scripted. Each Auditory Actor includes a script manager which maintains a set of time-stamped auditory events. Each auditory event represents an invocation of one of the Auditory Actor's interface methods. The Auditory World maintains a global clock which is used by the script manager to invoke the auditory events at their assigned time.

The VAS *Listener* is a specialized version of the Auditory Actor. The only added property is the head orientation of the user. This allows the Listener to be a sound producing entity, which can be a very useful feature for communicating information to the user such as motion cues (footsteps for example) and collision sounds.

The *Auditory Space* is another specialized Auditory Actor. It models distinct sonic spaces within the environment. An Auditory Space consists of one or more barriers each modeling an occluding surface. A barrier is essentially a bounded plane with which we associated a reflectivity and attenuation value. Ambient sounds which are attached to Auditory Spaces help distinguish sonic regions by giving them a unique character.

## 3.2. Sound Sources

The VAS system supports both sampled and synthetic sound sources. Sampled sounds are digital recordings of sounds that are temporally mapped and played back on demand. Like image-based texture maps in graphics, sampled sounds have gained wide popularity because they can be used to easily create complex sounds that would otherwise be difficult or impossible to synthesize. There are, however, drawbacks to this representation - the control parameters available to modify the characteristics of sampled sounds are limited due to the static nature of the representation.

Synthetic sounds are procedurally defined, one dimensional functions in the temporal domain. They are equivalent to procedural textures in the visual realm and have the same advantages and disadvantages. Their representation is inherently compact which makes them appealing for distributed applications such as VRML [8] clients. Their representation can be arbitrarily parameterized which makes them useful for data sonification (mapping data to sounds) and for synthesizing sounds from physical parameters. Finally, a procedural representation is a powerful construct giving the sound designer complete control over the shaping of the sound's timbre.

The primary disadvantage of synthetic sounds is that they are difficult to specify. They require a deep understanding of the sound synthesis process and an intuitive sense of how to achieve desired results. Another drawback of synthetic sounds is that they are computationally expensive. We have attempted to address both of these problems in our chosen representation of synthetic sounds, and in the graceful degradation approach implemented by the VAS scheduler.

Synthetic sounds are modeled using *Timbre Trees* [9], a functional representation of sound which was originally designed for use in computer animation. Timbre Trees represent sounds using a tree structure where the internal nodes of the tree are signal processing nodes, and the leaf nodes of the tree are signal generating nodes. Any node in the tree can be parameterized simply by inserting a named parameter in the node's definition. A parameter's value can be modified at run time through the Auditory Actor's interface. Our intention in using this representation is to develop

a library of parameterized trees where each tree represents a class of sounds. Specific results can then be achieved by varying a small set of intuitive parameters associated with each class of sounds.

VAS models sound sources as active objects so that each Sound object has a thread of execution associated with it. This thread is responsible for evaluating the sound signal and writing the resultant samples to its attached Device.

## 4. Managing Real-time Sound Generation

Given limited computational resources, we are faced with the eventuality of exceeding those limits as the number of active sounds in the system increases. We have developed a new approach in the VAS scheduler which prioritizes the sounds in the environment and enforces level-of-detail management based on this prioritization. A real-time scheduling algorithm was devised to manage the evaluation of concurrently active sounds in the server.

A real-time scheduling approach is generally necessary when the correctness of a computation not only depends on the results produced but also on the time at which they were produced. Clearly, the sound evaluation process falls under these constraints. The computation of a sample block by a Sound object must be completed and written to the Device before it completes the playback of the previously written sample block. Otherwise, intermittent breaks occur in the sound's playback.

In the study of different real-time scheduling strategies, workload models are used to describe the characteristics of the processes to be executed. In those models, an independent unit of computation is referred to as a *job*. Jobs are classified as *hard* or *soft* real-time. Hard real-time jobs are those where the results must be completed by a deadline or they are considered in error. For soft real-time jobs, this restriction is relaxed where the validity of the results decreases gradually as the deadline passes. Jobs are further classified as *periodic* when their execution consists of a periodic sequence of identical requests for execution termed *tasks*. The rate at which tasks are submitted for execution is the job's *period.*

We adopt the hard real-time periodic workload model for scheduling the execution of sound evaluation routines. In this workload model, a job set, J = {Jk}, consists of a set of tasks Tk,j for j = 1, 2, 3,.... The start time k,j of each task is the time before which its execution cannot begin. The period of a job Jk is k,j + 1 - k,j. The deadline of each task Tk,j is the start time of task Tk, j+1. The scheduling problem can then be stated as follows: given a job set J, each task Tk,j must be scheduled for execution such that it begins execution at some point at or after its start time k,j and completes its execution before k,j + 1. A schedule where these requirements are met is termed a precise schedule [7].

Scheduling concurrent sounds can be easily expressed in terms of the hard real-time periodic workload model. The job set consists of the active sounds in the environment. Each Sound object periodically submits requests for the evaluation of a sample block of fixed size. Because the block size is fixed for all sounds, the periods of all the jobs in the system are identical. The computation of a sample block must be completed in a time not to exceed this period. A precise schedule is achieved when all the active sounds complete the evaluation of their respective sample blocks before their deadline. This may not be possible if the computational resources available are exceeded. In order to address this problem we make use of a graceful degradation scheme to maintain real-time evaluation rates.

In a seminal paper Chung et.al [11] describe a technique for evaluating monotone processes using a model for imprecise computations. A monotone process is one which is guaranteed to produce increasingly accurate results as it is allowed to execute longer. The imprecise model partitions a task into a mandatory part and an optional part. The mandatory part is that required to produce results at the minimum acceptable precision. The mandatory task set is scheduled as a hard real-time task set and a precise schedule is obtained. Any remaining time in the schedule is used to execute the optional parts of the task set. The resultant schedule is termed a feasible schedule. The evaluation of a synthetic sound in real-time fits the imprecise computation model very well. Because evaluating the sound signal at successively higher sampling rates will increasingly produce better results, the evaluation routine is a monotone process.

The VAS scheduler extends this algorithm in two ways. The first is to incorporate a dynamic priority scheme. This is necessary since the relative importance of the sounds in the environment changes dynamically. The second is to incorporate a two tiered strategy for essential and non-essential sounds in the environment. The programmer specifies whether or not a sound is essential at its instantiation. This gives the programmer a measure of control over the scheduling algorithm. Essential sounds are scheduled as hard real-time jobs, while non-essential sounds are scheduled as soft real-time jobs, executing only if all the essential sounds have been evaluated at full resolution. Two components were necessary to incorporate this strategy into the VAS scheduler. One was a dynamic priority algorithm for rating the active sounds in the environment. The other was a mechanism for iteratively evaluating synthetic sounds. We discuss each of these and then present the scheduling algorithm.

## 4.1. Prioritizing Sounds

Ideally, what the priority algorithm must do is determine which sounds in the environment the listener is attending to. This, of course, is impossible to do specifically since attention is subjective; the listener cognitively decides what to pay attention to. The best we can do is to attempt to guess what the listener may be paying attention to based on the state of the listener and the sounds in the environment. Based on psychoacoustic principles, we have determined three factors which we use to rate sounds in the environment: the listener's gaze, the intensity of the sound, and the age of the sound.

The orienting response [12] is a human response to aural stimuli where the listener will attempt to support the perception of aural stimuli through visual correspondence. This reaction can be very useful in determining what the listener is attending to. Our priority algorithm approximates this reaction by weighing a sound using a cos  scaling factor where  is the angle between the listener's gaze vector and the sound (fig 3).

The intensity of a sound is important due to masking phenomenon [13]. A higher intensity sound will tend to mask a lower intensity sound if the two sounds are within the same frequency band. Determining the frequency content of a sound, however, requires an expensive Fourier transform which we opted against. Instead, we approximate this factor by giving louder sounds a higher scale factor.

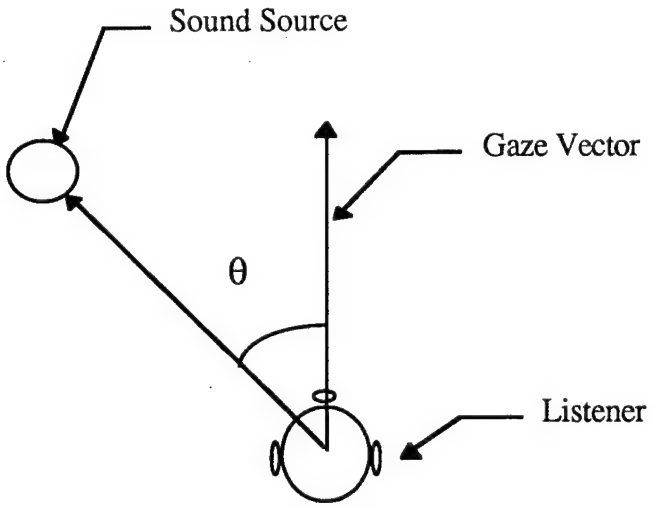


**Figure 3:** Measuring the angle between the listener's gaze vector and the sound source

The final scaling factor is based on the adaptation response of the human aural system [14]. Our sensitivity to aural stimuli decreases as the presence of the stimuli persists. This process continues for approximately three minutes after which it levels off. The scaling factor for this component begins at the start of a sound at its maximum level and decreases linearly until it reaches zero at the end three minutes.

Given these three scalars, the priority of a sound is calculated as follows:

$$P(s_i) = \begin{cases} W_g \cos\theta_i + W_t(3 - T_i) + W_l I_i & \text{if } 0 \le T_i \le 3 \\ W_g \cos\theta_i + W_l I_i & \text{otherwise} \end{cases}$$

where: $\theta_i$ is the angle shown in fig. 3

$I_i$ is the intensity of the sound at the listner

$T_i$ is the amount of time in seconds that a sound has been playing

$W_g$, $W_t$, $W_l$ are weights used to vary the contribution of each component

## 4.2. Iterative Evaluation of Synthetic Sounds

Sound objects evaluate samples in blocks corresponding to 100 ms at the global sampling rate of the system which is set upon invocation of the server. The size of the sample blocks is important because it determines the granularity of the parallelism among active sounds, and the latency between a call to start a sound and the appearance of the sound at the output device. If the size of the sample block is too small, the parallelism becomes too fine grained and scheduling overhead costs become prohibitive. A large sample block, on the other hand, increases the latency exhibited by the system where it becomes perceptible by the listener. The largest non-perceptible delay between a sound producing event and the occurrence of the corresponding sound is 100 ms. A block size corresponding to this value was chosen to maximize block size without introducing perceptible delay.

The imprecise computation model requires that we iteratively evaluate a Timbre Tree at successively higher resolutions. If the iteration is stopped before full resolution is reached, the evaluation routine will not have generated all the samples. The missing samples must be calculated using interpolation. To facilitate this, a buffer structure was devised which we call the *Interpolating Buffer* or *IBuffer*. The IBuffer consists of a set of mini-buffers in a stacked configuration as depicted in fig. 4. The depth of this stack determines the number of iterations necessary before full resolution is reached. On each iteration, the evaluation routine calculates a block of samples equivalent to 100 ms at 1/d of the global sampling rate, where d is the depth of the buffer. The resultant sample block is written to the IBuffer which fills one of its mini-buffers with the samples.

The order in which the mini-buffers are filled is predetermined by the IBuffer to minimize the number of empty mini-buffers between any two full mini-buffers. The starting time of the evaluation at each iteration is offset by an amount

$$\frac{1}{Rate} \times Depth$$

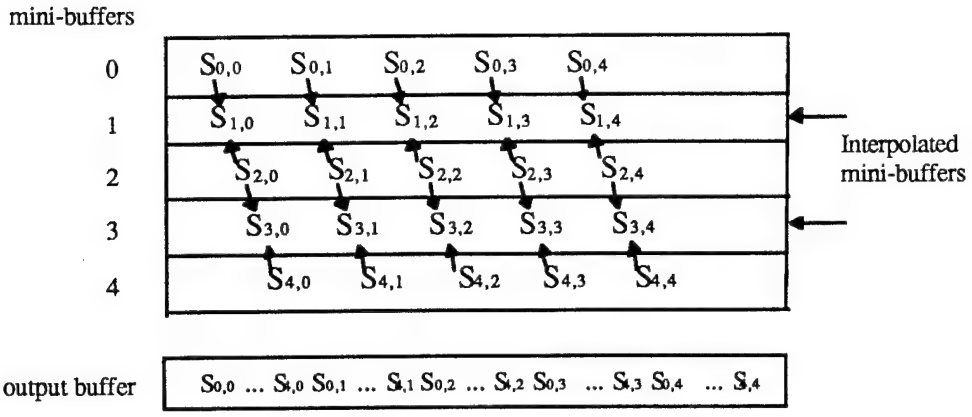


**Figure 5**: The interpolation step and output buffer. The IBuffer is depicted after three iterations. The samples in mini-buffers 0, 2, and 4 are evaluated. Those in 1 and 3 are interpolated.

## 4.3. Scheduling Algorithm

Scheduling algorithms which have been formulated to assign k prioritized, periodic jobs to n processors have unacceptable worst case conditions [15]. Instead, the problem is partitioned into two parts. First jobs are assigned to processors once and for all. Each processor is then scheduled as based on uniprocessor scheduling algorithms. The problem then becomes finding an optimal assignment of jobs to processors [16, 17]. The VAS scheduler uses a simple O(n) algorithm based on the greedy method [18]. The algorithm attempts to maintain an even load distribution over all the processors in order to minimize the total error exhibited by the system. When a job enters the scheduler, the processor list is searched for the processor with the least utilization factor and the job is assigned to that processor. The utilization factor of a processor is based on total execution time of the jobs already assigned to it.

Having assigned each processor a task set, the scheduler must determine the amount of processing time to assign to the optional part of each task. Processing times are chosen in order to minimize the error function $E$, while taking into account aesthetic constraints.

```
Initialize time = period, ready_set = { }, TE = { essential tasks}, TN = { non-essential tasks}, T =
TE

Step    for all t in T
                if  time >= t.iteration_time
                        assign t one iteration
                        time = time - t.iteration_time
                        add t to ready_set

        if time > 0
                Ptotal  =  t.priority for all t in ready_set
                for all t in ready_set  T
                        n = time *  t.priority/Ptotal
                        assign t n iterations
                        time = time - n * t.iteration_time
Iterate if time > 0
        T = TN, repeat Step
```

**Figure 6:** VAS's scheduling algorithm

$$E = \sum_{k=1}^{K} w_k (1 - \frac{\sigma_k}{\tau_k})$$

where $w_k$ is the relative importance of task k

$\sigma_k$ is the amount of time assigned to task k

$\tau_k$ is the time required to complete task k at full resolution

Because the human ear is very sensitive to discontinuities in the timbre of sounds, degradation must occur gradually as load conditions change if perceptible effects are to be minimized. Furthermore, because the listener can be cognitively attending to multiple sounds occurring within the environment, the scheduler must adhere to a fair strategy avoiding the starvation of less important sounds.

The VAS scheduler uses a heuristic algorithm which minimizes the error function $E$ according to the above constraints. In order to integrate soft real-time jobs into the job mix, the

VAS scheduler maintains two lists of active tasks: one for essential sounds and the other for non-essential sounds. Non-essential tasks are only scheduled once all the essential tasks have been assigned their full execution time.

The scheduling algorithm (fig. 6) minimizes the error function while observing aesthetic constraints. If a precise schedule is possible then each task will execute at least one iteration. If conditions deteriorate to the point where a precise schedule of one iteration is not be possible, the algorithm schedules as many of the highest priority tasks as possible, dropping the lower priority tasks from the schedule.

## 5. Real-time Sound Rendering

In designing VAS, we placed significant emphasis on having the system support a variety of rendering techniques. An abstract Device object defines the interface between any rendering mechanism and VAS. New devices are added to VAS by specializing the Device object through inheritance. The specialized object implements the abstract device's primitives by interfacing with a an external physical device through its supplied programming interface or by implementing them directly in software.

This architecture, while general, requires that the spatialization device accept multiple sample streams to be rendered and combined in some fashion depending on the rendering technique. Many hardware-based localization devices, however, either do not support this functionality, or only support a limited number of external inputs. Their architecture is optimized for sound sources consisting of sampled sounds which reside locally in the device. This scheme has the advantage of off-loading all the audio processing onto the hardware device. However, it constrains the available audio processing to whatever the hardware device supports, limiting the user's options and requiring expensive hardware upgrades to increase the functionality of the available audio processing.

As an alternative to hardware-based spatialization, VAS provides a no cost spatialization option using the audio capabilities found in most modern computers. A specialized Device object

was created which uses the four channel output available on Silicon Graphics computers. Each Device object generates four channels of output properly scaled to position the sound to the desired location in the sonic environment. The spatialization algorithm determines the scaling for each channel.

## 6. Conclusion

The VAS server addresses important problems which are currently limiting the use of sound in VEs. High level abstractions are provided for modeling the auditory world facilitating the integration of sound into VE systems. VAS's synthetic sound sources make it an ideal test bed for studying sound synthesis from motion parameters as well as data sonification problems. An extensible architecture allows VAS to provide a variety of rendering options including a no-cost spatialization option. Finally, VAS frees the programmer from dealing with the contingency of exceeding a limit on the number of current sounds possible through its use of a novel graceful degradation scheme for sound generation. While this technique was specifically designed for the sound generation process, the results presented may be applicable to other problems, particularly the real-time generation of procedural textures.

## 7. References


[1] S. Das, T. DeFanti, D. Sandin, *An Organization for High-Level Interactive Control of Sound*, Proceedings of the Second International Conference on Auditory Displays, ICAD '94, Nov 1994, pp.203-216

[2] Burgess, D. A., Verlinden, J. C., (1193) *An Architecture for Spatial Audio Servers*, GVU Technical Report GIT-GVU-93-34

[3] S. H. Foster, E. M. Wenzel, and R. M. Taylor, *Real Time Synthesis of Complex Acoustic Environments*, Proc. CHI'91, ACM Conference on Computer Human Interaction, New Orleans, LA, April 1991

[4] Crystal River Engineering, Inc., *Acoustetron II Operation Manual*, Crystal River Engineering, Inc., 1995

[5] S. L. Russell, *NPSNET-3D Sound Server: An Effective User of the Auditory Channel*, Master's Thesis, Naval Postgraduate School, Monterey, California, September, 1995

[6] Visual Synthesis, Inc. *Personal Audio*, Visual Synthesis, Inc. 1991

[7] J. A. Stankovic, *Misconceptions about Real-time Computing - A Serious Problem for Next-Generation Systems*, IEEE Computer, Vol. 21, No. 10, pp. 10-18, 1988

[8] M. Pesce, *VRML: Brousing and Buildng Cyberspace*, New Riders Publishing, 1995

[9] J. Hahn, J. Geigel, J. W. Lee, L. Gritz, T. Takala, and S. Mishra, *An Integrated Approach to Sound and Motion*, Journal of Visualization and Computer Animation, Vol. 6, No. 2, pp. 109-123

[10] T. Takala, J. Hahn, *Sound Rendering*, Computer Graphics, Vol.26, No 2, July 1992, pp.211-220

[11] J. Y. Chung, J.W.S Liu, and K. J. Lin, *Scheduling Real-Time, Periodic Jobs Using Imprecise Results*, Proc IEEE RTS, 1987

[12] S. Coren, C. Porac, and L. M. Ward, *Sensation and Perception 2nd ed.*, Academic Press, Orlando, Florida, 1984

[13] S. Handel, *Listening*, The MIT Press, Cambridge Massachusetts, 1989

[14] J. V. Tobias, *Foundation of Modern Auditory Thoery*, vol. 1, Academic Press, NY and London, 1970

[15] S. K. Dhall and C. L. Liu, *On a real-time scheduling problem*, Oper. Res., vol 26, no. 1, pp.127-140, 1978

[16] C. L. Lui and J. W. Layland, *Scheduling algorithms for multiprogramming in a hard real-time environment*, J. ACM, vol. 20, no. 1, pp. 46-61, Jan. 1973

[17] J. Y. Chung, J. W. S. Lui and K. J. Lin, *Scheduling periodic jobs that allow imprecise results*, IEEE Transactions, vol. 39, no. 9, pp. 1156-1174, 1990

[18] E. Horowitz and S. Sahani, *Fundamentals of Computer Algorithms*, Computer Science Press, 1978

[19] M. Beaudouin-Lafon, W. W. Gaver, *ENO: Synthesizing Structured Sound Spaces,* Proc. ACM Symposium on User Interface Software and Technology (UIST'94, Marina del Ray, CA, November 1994), pp 49-56, ACM Press, New York, 1994